Vivekanand Education Society's
Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai

Artificial Intelligence and Data Science Department

Deep Learning/Odd Sem 2023-23/Experiment 3a

Name: Rupesh Dhirwani	**Class/Roll No. :** D16AD - 10	**Grade :**

Title of Experiment: Autoencoder for image compression

Objective of Experiment: The objective of this experiment is to design and implement an autoencoder based image compression system that can effectively reduce the size of an input image while preserving their essential visual information. This system aims to explore the potential of autoencoders in the field of image compression, leveraging the capabilities of learning compact representation of images.

Outcome of Experiment: Thus we have implemented an autoencoder model trained, which has learnt to encode and decode images effectively.

Problem Statement: Traditional image compression techniques such as JPEG, often result in loss of image quality due to lossy image compression algorithms. The problem is to develop an autoencoder system that addresses the issue of compressing the images into a lower dimensional representation while minimizing the compression ratio and image quality, making it suitable for various applications such as efficient storage, transmission, sharing of files or images while maintaining their perpetual fidelity.

Description / Theory :

An autoencoder is a type of artificial neural network used in machine learning, deep learning and for various other applications including image compression. The main idea behind using autoencoders for image compression is to learn a compact representation of an input image while preserving its essential features. Here's a short description for how autoencoders are applied to image compression:

Architecture:

An autoencoder consists of two main parts: an encoder and a decoder.

Encoder:

The encoder takes an input image and maps it to a lower-dimensional representation called the "latent space" or "encoding." This encoding typically has fewer dimensions than the original image, which leads to compression.

Decoder. The decoder takes the encoded representation and attempts to reconstruct the original image from it.

Training:

Autoencoders are trained using unsupervised learning. The training objective is to minimize the reconstruction error, which measures how well the decoder can reconstruct the input image from its encoding.

Common loss functions for image compression tasks include mean squared error (MSE) or binary cross-entropy, depending on whether the images are continuous or binary (e.g., grayscale or black-and-white).

Compression:

The key to image compression with autoencoders is the reduction in the dimensionality of the encoding compared to the original image. By encoding the image in a lower-dimensional space, it's possible to represent the image using fewer bits, which results in compression.

Benefits:

Autoencoders offer lossy compression, meaning that some information is lost during compression, but the goal is to preserve the essential features for human perception. They can be used for various image compression applications, including reducing storage space and speeding up image transmission over networks

Trade-offs:

The trade-off in using autoencoders for compression is the balance between compression ratio and image quality. More aggressive compression may lead to greater loss of image quality. The choice of architecture, hyperparameters, and the size of the latent space can impact compression performance.

Applications:

Autoencoders for image compression are used in various fields, including medical imaging, video streaming, and image transmission in low-bandwidth environments. They are also used as a pre-processing step for other computer vision tasks, where reducing the dimensionality of images can improve efficiency.

Program:

```
import tensorflow

import tensorflow.keras.layers

import tensorflow.keras.models

import tensorflow.keras.optimizers

import tensorflow.keras.datasets

import numpy

import matplotlib.pyplot

x = tensorflow.keras.layers.Input(shape= (784), name = "encoder_input")


encoder_dense_layer1 = tensorflow.keras.layers.Dense(units = 300, name =
"encoder_dense_1")(x)
```

encoder_activ_layer1 = tensorflow.keras.layers.LeakyReLU(name = "encoder_leakyrelu_1")(encoder_dense_layer1)

encoder_dense_layer2 = tensorflow.keras.layers.Dense(units = 2, name = "encoder_dense_2")(encoder_activ_layer1)

encoder_output = tensorflow.keras.layers.LeakyReLU(name = "encoder_output")(encoder_dense_layer2)

encoder = tensorflow.keras.models.Model(x, encoder_output, name = "encoder_model")

encoder.summary()

decoder_input = tensorflow.keras.layers.Input(shape = (2), name = "decoder_input")

decoder_dense_layer1 = tensorflow.keras.layers.Dense(units = 300, name = "decoder_dense_1")(decoder_input)

decoder_activ_layer1 = tensorflow.keras.layers.LeakyReLU(name = "decoder_leakyrelu_1")(decoder_dense_layer1)

decoder_dense_layer2 = tensorflow.keras.layers.Dense(units = 784, name = "decoder_dense_2")(decoder_activ_layer1)

decoder_output = tensorflow.keras.layers.LeakyReLU(name = "decoder_output")(decoder_dense_layer2)

decoder = tensorflow.keras.models.Model(decoder_input,decoder_output,name="decoder_model")

decoder.summary()

Auto encoder

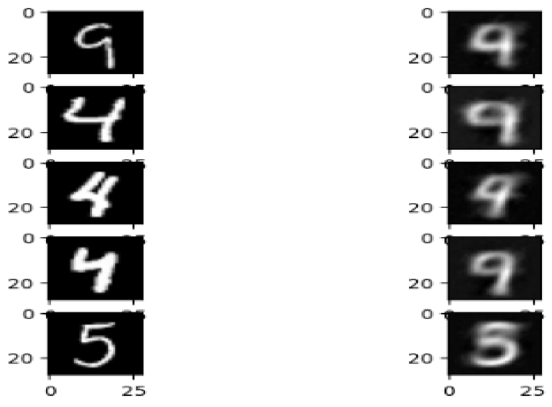
ae_input = tensorflow.keras.layers.Input(shape=(784), name = "AE_input")

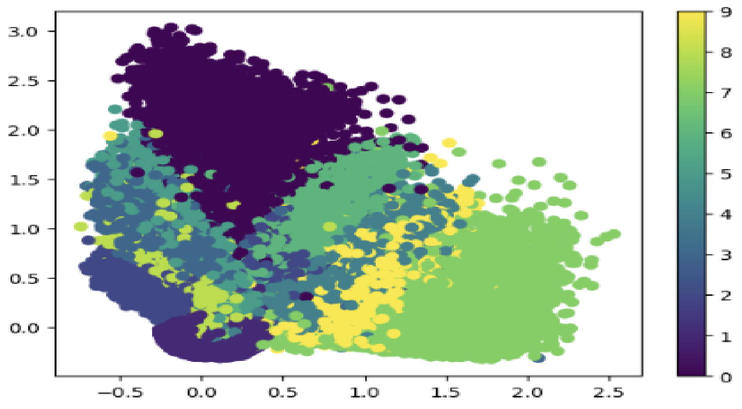
ae_encoder_output = encoder(ae_input)

ae_decoder_output = decoder(ae_encoder_output)

```
ae = tensorflow.keras.models.Model(ae_input, ae_decoder_output, name="AE")

ae.summary()

def rmse(y_true, y_predict):

    return tensorflow.keras.backend.mean(tensorflow.keras.backend.square(y_true, y_predict))

ae.compile(loss="mse", optimizer = tensorflow.keras.optimizers.Adam(learning_rate=0.0005))

(x_train_orig, y_train), (x_test_orig, y_test) = tensorflow.keras.datasets.mnist.load_data()

x_train_orig = x_train_orig.astype("float32") / 2555.0

x_test_orig = x_test_orig.astype("float32") / 2555.0

x_train = numpy.reshape(x_train_orig, newshape = (x_train_orig.shape[0],
numpy.prod(x_train_orig.shape[1:])))

x_test = numpy.reshape(x_test_orig, newshape = (x_test_orig.shape[0],
numpy.prod(x_test_orig.shape[1:])))

ae.fit(x_train, x_train, epochs = 20, batch_size = 256, shuffle=True, validation_data=(x_test,
x_test))

encoded_images = encoder.predict(x_train)

decoded_images = decoder.predict(encoded_images)

decoded_images_orig = numpy.reshape(decoded_images,
newshape=(decoded_images.shape[0],28,28))

num_images_to_show = 5

for im_ind in range(num_images_to_show):

    plot_ind = im_ind * 2 +1

    rand_ind = numpy.random.randint(low = 0, high = x_train.shape[0])

    matplotlib.pyplot.subplot(num_images_to_show,2,plot_ind)
    matplotlib.pyplot.imshow(x_train_orig[rand_ind, :, :], cmap="gray")

    matplotlib.pyplot.subplot(num_images_to_show,2,plot_ind+1)

    matplotlib.pyplot.imshow(decoded_images_orig[rand_ind, :, :], cmap="gray")


matplotlib.pyplot.figure()

matplotlib.pyplot.scatter(encoded_images[:,0], encoded_images[:,1], c=y_train)

matplotlib.pyplot.colorbar()
```

OUTPUT:





Results:

Dataset Loading and Preprocessing:

The MNIST dataset, consisting of grayscale hand-written digits, is loaded. The pixel values of the images are scaled between 0 and I by dividing by 255.0.The data is reshaped into flat vectors of size 784 (28x28).

Autoencoder Architecture:

The autoencoder architecture is defined with an input layer of 784 neurons (the flattened image).A dense layer with 128 neurons and ReLU activation serves as the encoder. Another dense layer with 784 neurons and sigmoid activation serves as the decoder. The autoencoder model is created, which maps the input to its reconstruction.

Training:

The model is compiled with the Adam optimizer and binary cross-entropy loss function. The training process involves 5 epochs with a batch size of 28, and the data is shuffled during training.Both the training and validation datasets are the same (x train and x test).

Encoding and Decoding

The trained autoencoder is used to encode and decode the images from the test dataset. The encoded images represent a compressed form of the input data. The decoded images are the model's attempt to reconstruct the original images from their encoded representations.

Visualization:

The program generates a visual comparison of original, encoded, and decoded images for 10 samples from the test dataset. Three rows of images are displayed: the top row for original images, the middle row for encoded images, and the bottom row for decoded images.

Discussion:

The autoencoder successfully learns a compressed presentation of the input images in its encoded layer. The encoded images capture the important features of the original images, albeit with a lower dimensionality. The decoded images show that the model can reconstruct the input data with reasonable accuracy, although there may be some loss of fine details. This type of autoencoder can be used for various applications, including image denoising, dimensionality reduction, and feature extraction.